The Basics of UNIX/Linux

14. File I/O

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU

datalab

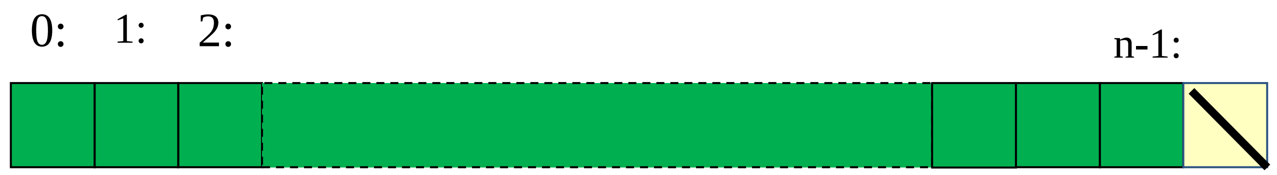
data science laboratory

Outline

- **File in C**
- Text File I/O
- Binary File I/O

File

- C views each file as a sequence of bytes
  - File ends with the end-of-file marker
  - Or, file ends at a specified byte



- A file
  - has a name
  - The data on a file has a format

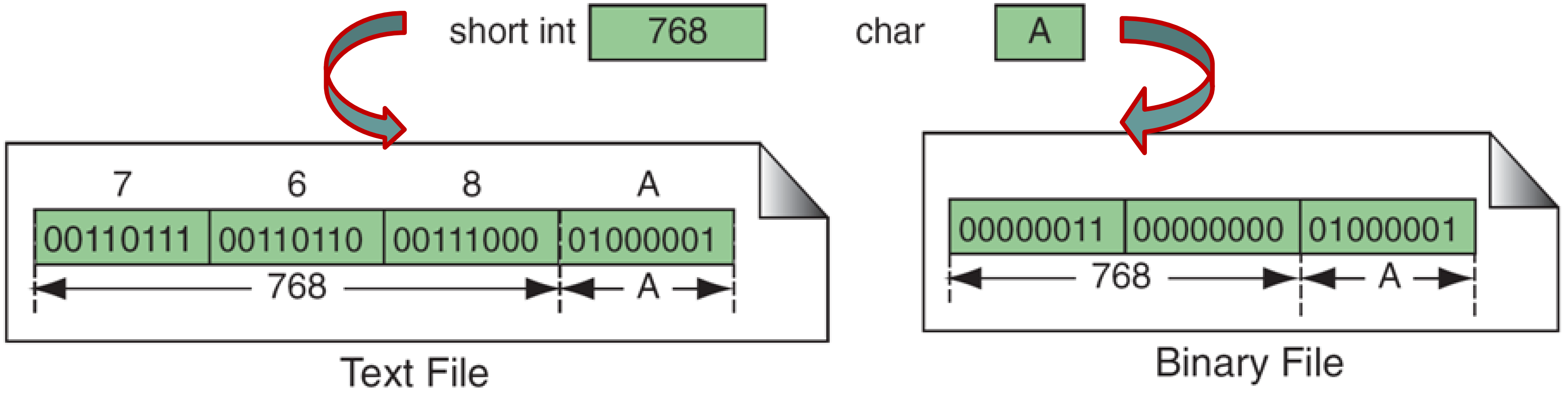
→ We can read/write a file if we know its name and format

Files in C

- #include <stdio.h>
- **FILE** object contains file stream information
- Special files defined in stdio:
  - stdin: Standard input
  - stdout: Standard ouput
  - stderr: Standard error
- EOF: end-of-file, a special negative integer constant

Text and Binary Files

- Text file
  - A file that contains characters from the ASCII or Unicode character sets
- Binary file
  - A file that contains data in a specific format, requiring special interpretation of its bits

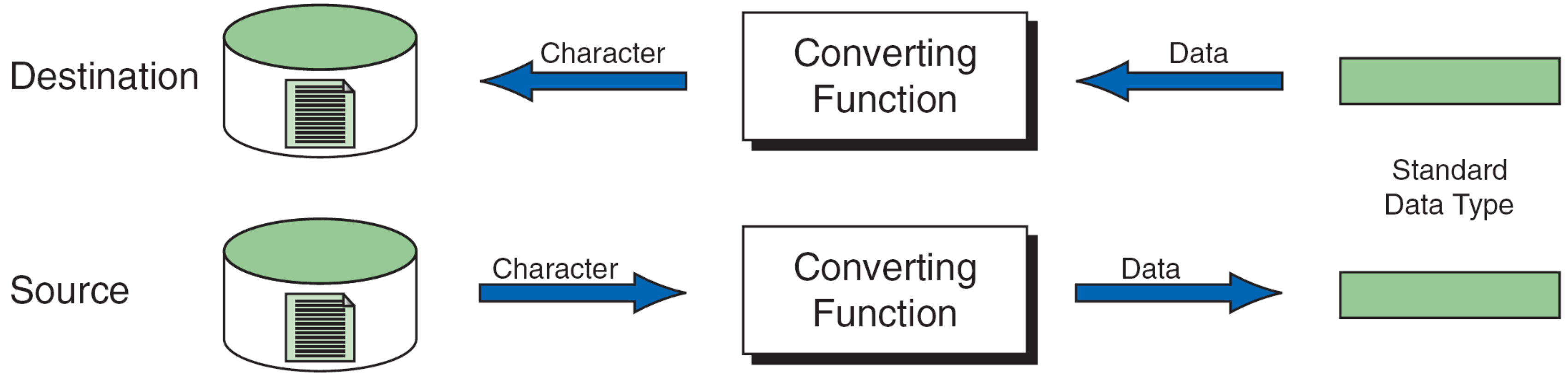


Reading and Writing Files

- To read a file
  - We must know its name
  - We must open it (for reading)
  - Then we can read
  - Then we must close it
    - That is typically done implicitly

- To write a file
  - We must name it
  - We must open it (for writing)
    - Or create a new file of that name
  - Then we can write it
  - We must close it
    - That is typically done implicitly

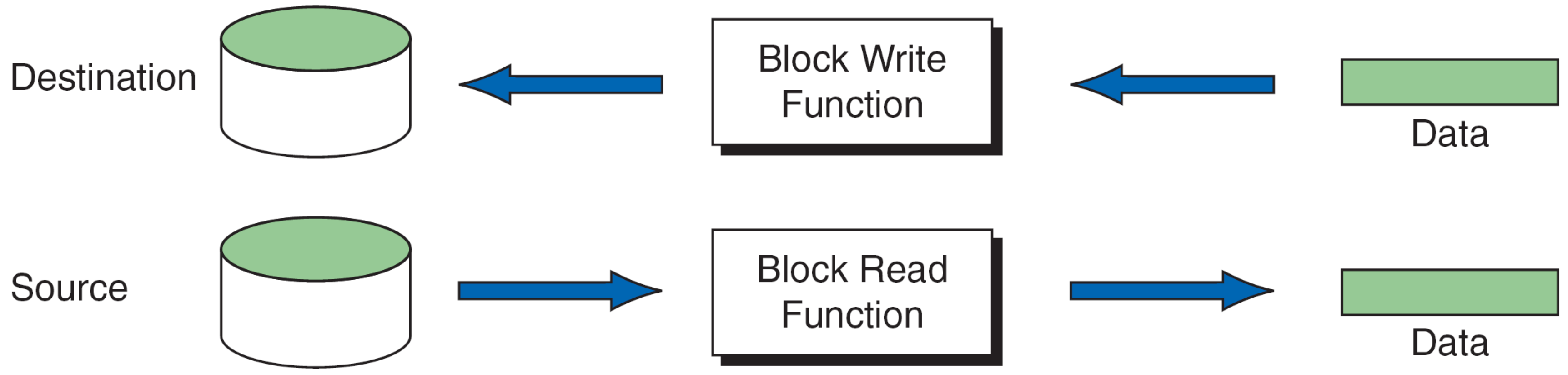
Reading and Writing Text Files



Formatted input/output, character input/output, and string input/output functions can be used only with text files.

Block Input and Output

- For Binary Files



Types of I/O functions

- Character-based
  - getc( ), fgetc( ); putc( ), fputc( ), …
- Line-based
  - gets( ), fgets( ); puts( ), fputs( ), …
- formatted
  - scanf( ), fscanf( ); printf( ), fprintf( ) …
- Binary
  - fread( ), fwrite( ), …

Outline

- File in C
- **Text File I/O**
- Binary File I/O

Opening/Closing binary files

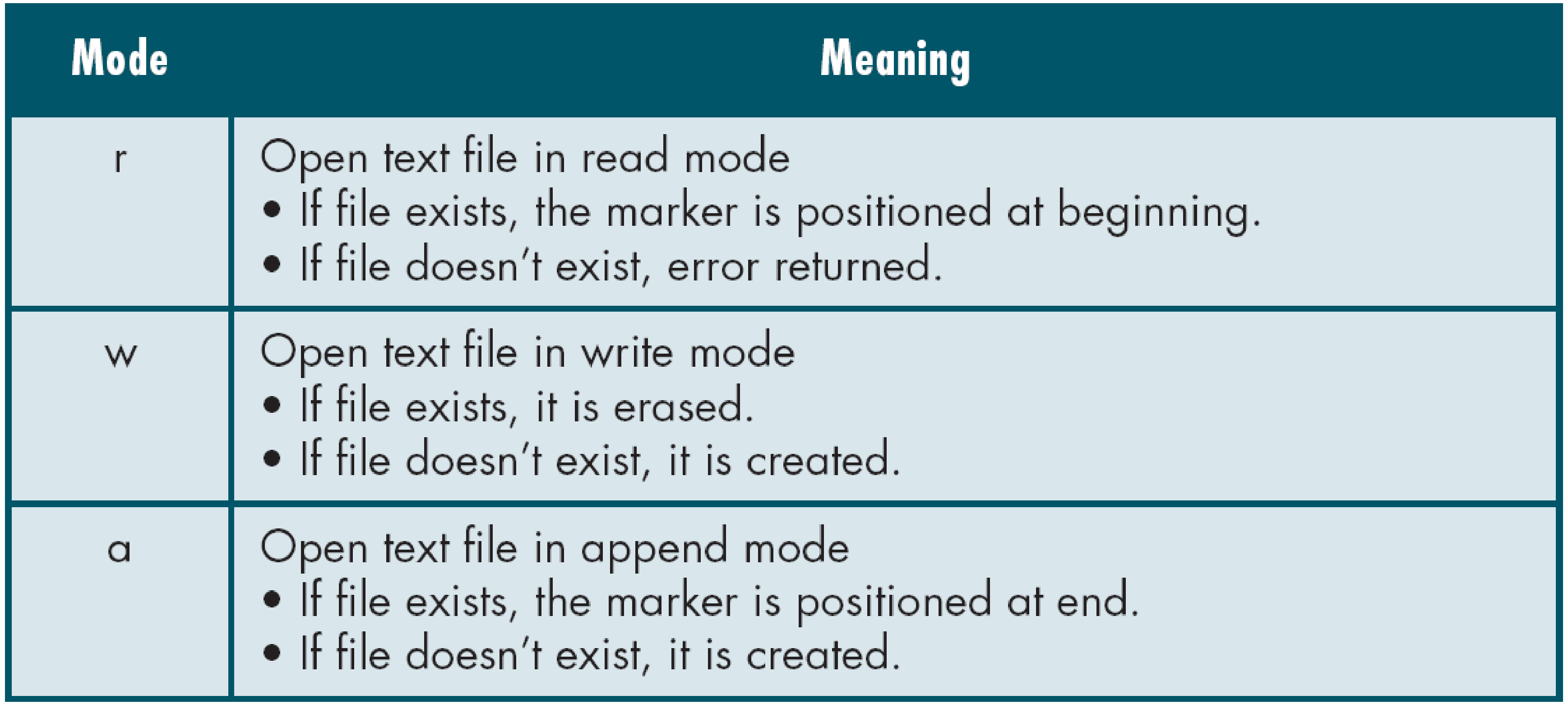
```
FILE *fopen(const char *filename, const char *mode);
```

- Same as text I/O, but mode is different
- Mode
  - b: binary indicator
  - Six binary modes
    - read binary(**rb**), write binary(**wb**), append binary(**ab**), read and update binary(**r+b**), write and update binary(**w+b**), and append and update binary (**a+b**)

```
int fclose(FILE* fp);
```

- Same as text I/O

File Open Modes



Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

Character I/O functions (1/2)

```
#include <stdio.h>
int getc (FILE *fp );
int fgetc (FILE *fp );
int getchar (void );
```

- Reading a character from a file

```
#include <stdio.h>
int ungetc (int c, FILE *fp );
```

- Un-reading a character
  - Virtually puts a character back into the file
  - Doesn't modify the file
  - May be a different character than the last one read

Character I/O functions (2/2)

```
#include <stdio.h>
int putc (int c, FILE *fp );
int fputc (int c, FILE *fp );
int putchar (int c );
```

- Writing a character to a file

Example1

- Copy1.c
  - Stdin and Stdout

```c
#include <stdio.h>

int main()
{
    int c;

    c = fgetc(stdin); // read ASCII code from the keyboard
    while (c != EOF)
    {
        fputc(c, stdout); // write a value of c into stdout file
        c = fgetc(stdin); // read a new char from the keyboard
    }
}
```

Example2: character-based I/O

- copy2.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *fp;
    int c;

    fp = fopen(argv[1], "w"); // write mode
    c = fgetc(stdin); // read ASCII code from the keyboard
    while (c != EOF)
    {
        putc(c, fp); // write a value of c into the file ponted by fp
        c = fgetc(stdin); // read a new char from the keyboard
    }
    printf("A write operation to %s is completed.\n", argv[1]);
}
```

Example3: File copy with a character IO

- Copy3.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char c;
    FILE *fp1, *fp2;

    fp1 = fopen(argv[1], "r"); // read mode
    if(fp1 == NULL)
    {
        printf("Error: Cannot open the file %s\n", argv[1]);
        exit(1);
    }
    fp2 = fopen(argv[2], "w"); // read ASCII code from the keyboard
    while ( (c= fgetc(fp1)) != EOF)
    {
        fputc(c, fp2); // write a value of c into the file pointed by fp
    }
    fclose(fp1);
    fclose(fp2);
    printf("Copy from %s to %s is completed.\n", argv[1], argv[2]);
}
```

Line-based I/O functions (1/2)

```
#include <stdio.h>

char *fgets (char  *buf, int  n, FILE *fp );

char *gets (char *buf );
```

- Reading a string from a file
  - Reads at most (num-1) characters from the stream into str
  - Null-terminates the string read (adds a '\0' to the end)
  - Stops after a newline character is read
  - Stops if the end of the file is encountered
    - Caveat: if no characters are read, str is not modified

Line-based I/O functions (2/2)

```
#include <stdio.h>
int fputs (const char  *str, FILE *fp );
int  puts (const char *str);
```

- OUTPUT / EFFECT
  - On success, writes the string to the file and returns a non-negative value
  - On failure, returns EOF and sets the error indicator

Example4: lineio.c

```c
#include <stdio.h>
#define MAXLINE 80

int main(int argc, char *argv[])
{
    FILE *fp;
    int line = 0;
    char buffer[MAXLINE];

    if (argc != 2)
    {
        fprintf(stderr, "Usage: line filename\n");
        return 1;
    }
    if ( (fp = fopen(argv[1],"r")) == NULL )
    {
        fprintf(stderr, "Usage: line filename\n");
        return 2;
    }
    while (fgets(buffer, MAXLINE, fp) != NULL)  // read one MAXLINE
    {
        line++;
        printf("%3d %s", line, buffer); // print with a line number
    }
    fclose(fp);
}
```

Formatted I/O functions

```
#include <stdio.h>
int fscanf ( FILE * stream, const char * format, ... )
```

- Reading formatted data from a file
  - Format string is analogous to **printf** format string

```
#include <stdio.h>
int fprintf ( FILE * stream, const char * format, ... )
```

- Writing a formatted string to a file
  - The format string is same as for **printf**

Example: fprint.c

student.h

```c
struct student
{
    int id;
    char name[20];
    double score;
};
```

```c
#include <stdio.h>
#include "student.h"

int main(int argc, char **argv)
{
    struct student record;
    FILE *fp;

    if (argc !=2)
    {
        fprintf(stderr,"Usage: %s filename\n", argv[0]);
        return 1;
    }
    fp = fopen(argv[1], "r");
    printf("%s %7s %6s\n", "Sno", "Sname", "Sgrade");
    while (fscanf(fp,"%d %s %lf", &record.id, record.name, &record.score) == 3)
        printf("%d %s %lf", record.id, record.name, record.score);
    printf("\n");
    fclose(fp);
    return 0;
}
```

Example: fscanf.c

student.h

```c
struct student
{
    int id;
    char name[20];
    double score;
};
```

```c
#include <stdio.h>
#include "student.h"

int main(int argc, char **argv)
{
    struct student record;
    FILE *fp;

    if (argc !=2)
    {
        fprintf(stderr,"Usage: %s filename\n", argv[0]);
        return 1;
    }
    fp = fopen(argv[1], "r");
    printf("%s %7s %6s\n", "Sno", "Sname", "Sgrade");
    while (fscanf(fp,"%d %s %lf", &record.id, record.name, &record.score) == 3)
        printf("%d %s %lf", record.id, record.name, record.score);
    printf("\n");
    fclose(fp);
    return 0;
}
```

Outline

- File in C
- Text File I/O
- **Binary File I/O**

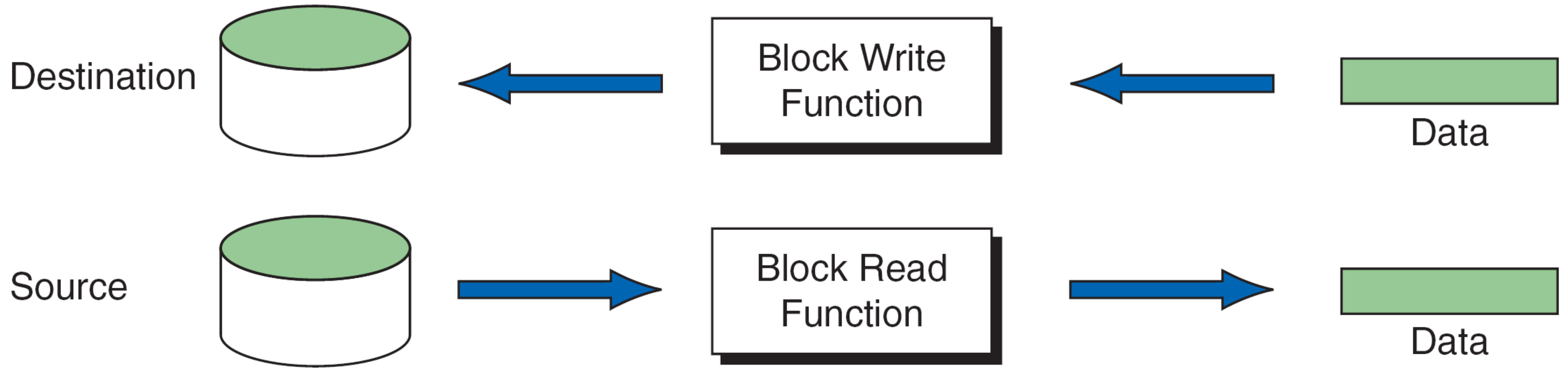
Necessity of Binary I/O

- read or write the entire C structure
  - Method1: getc( ), putc( )
    - Loop through the entire structure and process it one character at a time
  - Method2: fgets( ), fputs( )
    - fgets( ): Can not handle well when null or newline characters exists in the middle of a structure
    - fputs( ): Can not handle well when null characters exists in the middle of a structure
  - Method3:
    - Need functions that can read and write as much data as we want

  →Binary I/O functions

Basic concept

- Block Input and Output



  - Interprets any data as **consecutive bytes** and stores it in a file
  - Reads data stored in a file in a continuous byte format
    - Store the continuous bytes into the original variable

File Read/write Operation

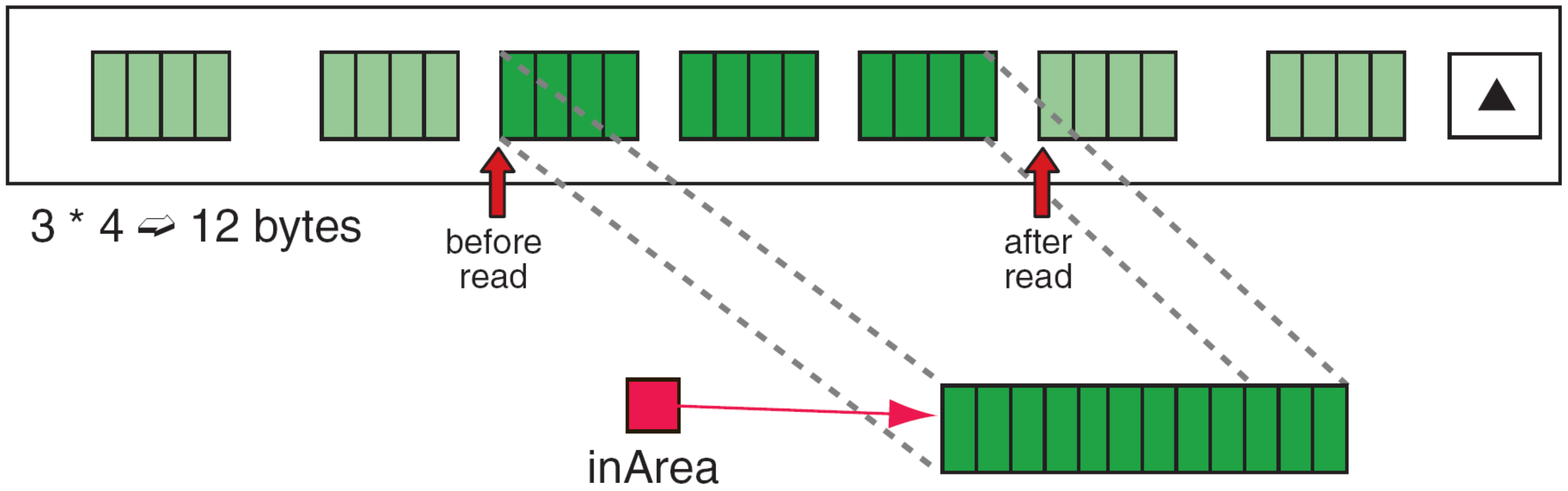
```
#include <stdio.h>
size_t fread (void *buffer, size_t  size, size_t  no_items, FILE  *fp );
size_t fwrite (const void *buffer, size_t  size, size_t  no_objs, FILE  *fp );
```

- fread()
  - Returns: actual number of items read; NULL if error or end of file.
  - Description: reads no_items, each of size, size, bytes from stream, fp, into buffer.

- fwrite()
  - returns: the number of objects written if successful; less than no_objs on error.
  - Description: writes (appends) no_objs objects of size, size, from buffer to stream.

File Read Operation

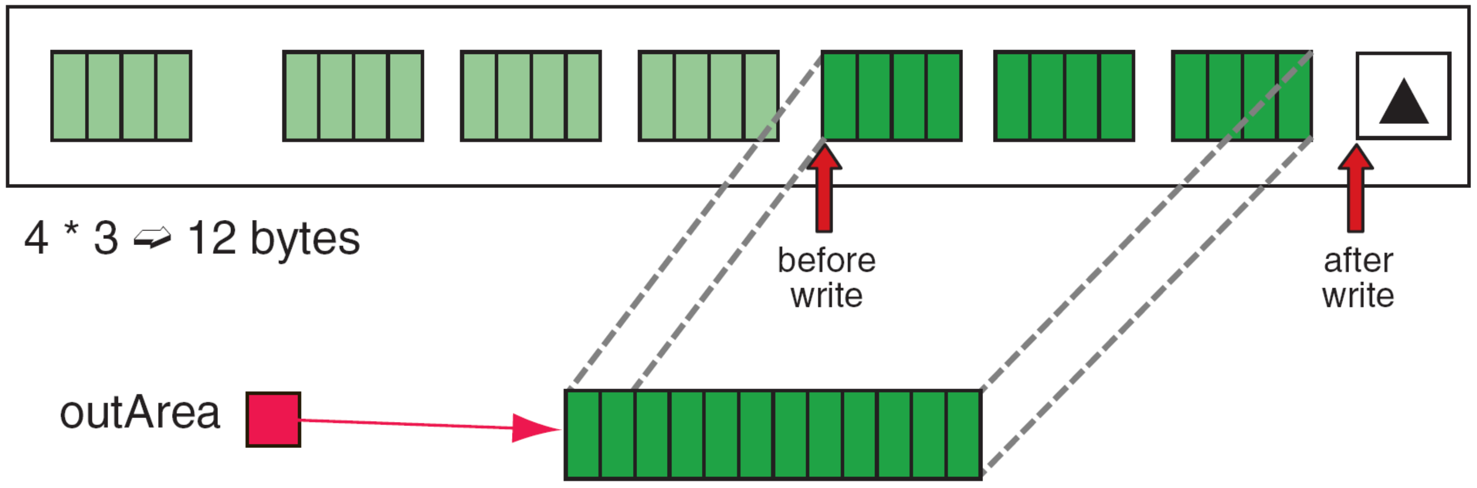
- Read binary data from file
  - int fread(void *pInArea, int elementSize, int count, FILE *sp);
  - Reads up to *count* objects, each *elementSize* bytes long, from input file *sp*, storing them in the memory pointed to by p*InArea*



```
fread (inArea, sizeof (int), 3, spData);
```

File Write Operation

- Write binary data to file
  - int fwrite(void *pOutArea, int elementSize, int count, FILE* sp);
  - Writes up to *count* objects, each *elementSize* bytes long, from the memory pointed to by *pOutArea* to the output file *sp*



```
fwrite (outArea, sizeof (int), 3, spOut);
```

Example: fwrite

```
#include <stdio.h>
#include <stdlib.h>
#include "student.h"

int main(int argc, char **argv)
{
    struct student record;
    FILE *fp;

    if (argc !=2)
    {
        fprintf(stderr,"Usage: %s filename\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    fp = fopen(argv[1], "wb");
    printf("%s %7s %6s\n", "Sno", "Sname", "Sgrade");
    while (scanf("%d %s %lf", &record.id, record.name, &record.score) == 3)
        fwrite(&record, sizeof(record), 1, fp);

    fclose(fp);
    return 0;
}
```

Example: fread (1/2)

```c
#include <stdio.h>
#include "student.h"

int main(int argc, char **argv)
{
    struct student record;
    FILE *fp;

    if (argc !=2)
    {
        fprintf(stderr,"Usage: %s filename\n", argv[0]);
        return 1;
    }

    if ( (fp = fopen(argv[1], "rb"))  == NULL)
    {
        fprintf(stderr, "Error: Cannot open the file %s\n", argv[1]);
        return 2;
    }
```

Example: fread (2/2)

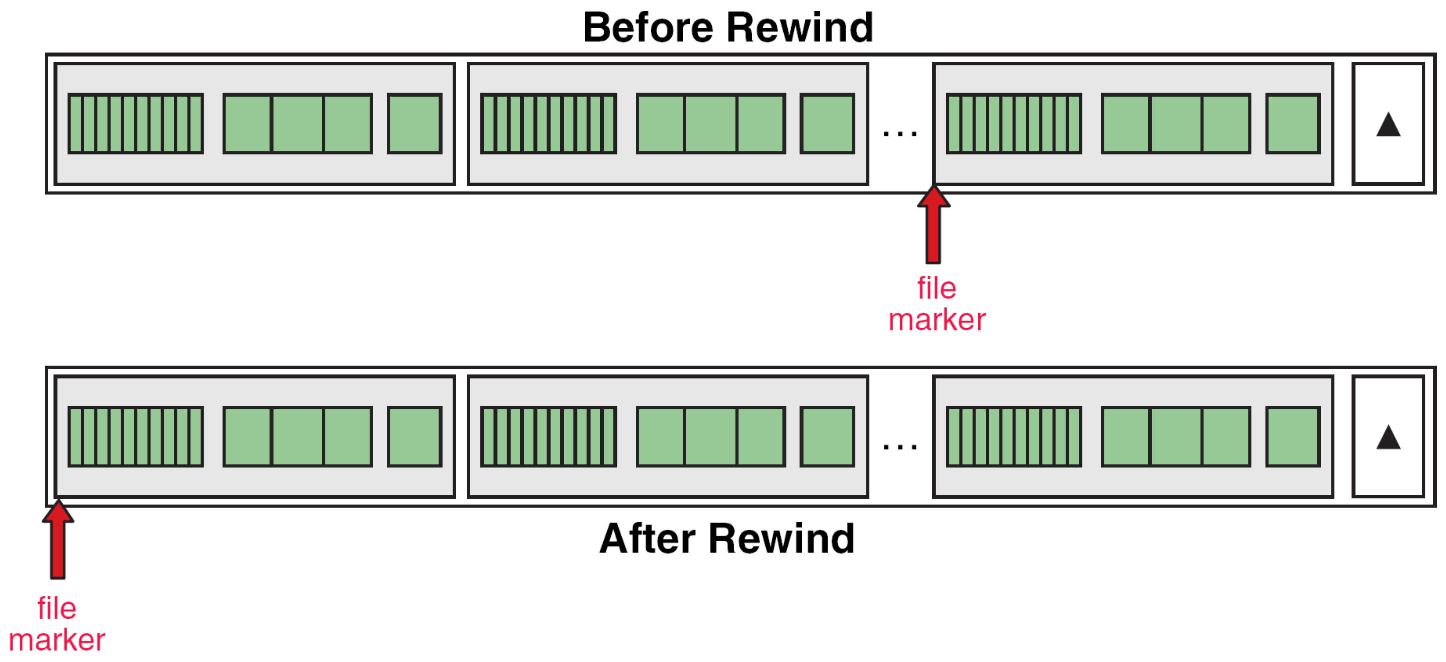
```c
    printf("-------------------------------------\n");
    printf("%s\t %7s\t %6s\n", "Sno", "Sname", "Sgrade");
    printf("-------------------------------------\n");

    while (fread(&record, sizeof(record), 1, fp) > 0)
    {
        if (record.id !=0)
            printf("%d\t %s\t %lf\n", record.id, record.name, record.score);
    }
    printf("\n");
    printf("-------------------------------------\n");
    return 0;
}
```

Positioning

- void rewind(FILE* stream);
- long int ftell(FILE* stream);
- int fseek(FILE* stream, long offset, int wherefrom);

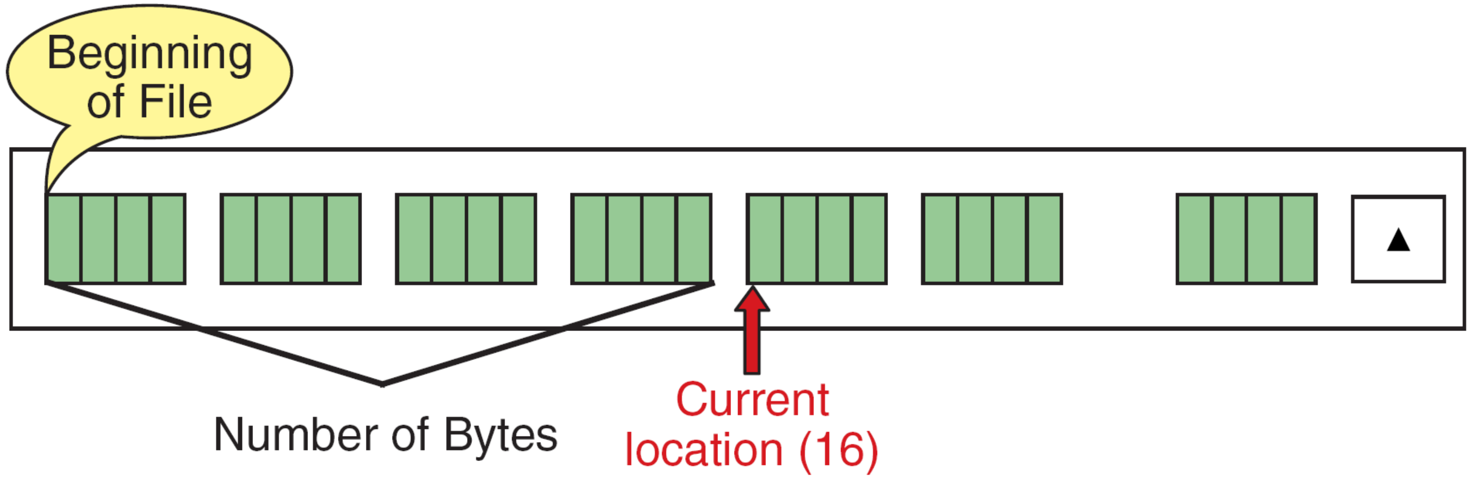
Rewind File

- Rewind
  - void rewind(FILE* stream);
  - Sets the file position indicator for *stream* to the beginning of the file



Current Location (*ftell*) Operation

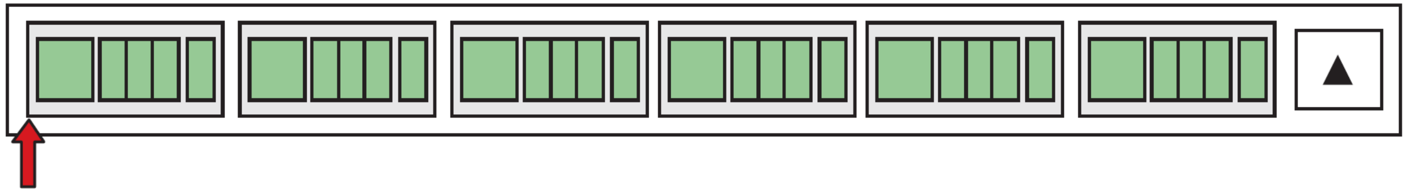
- Current location
  - long int ftell(FILE* stream);
  - Returns the current file position for *stream*
  - Returned value is the number of bytes from the beginning of the file to the current file position

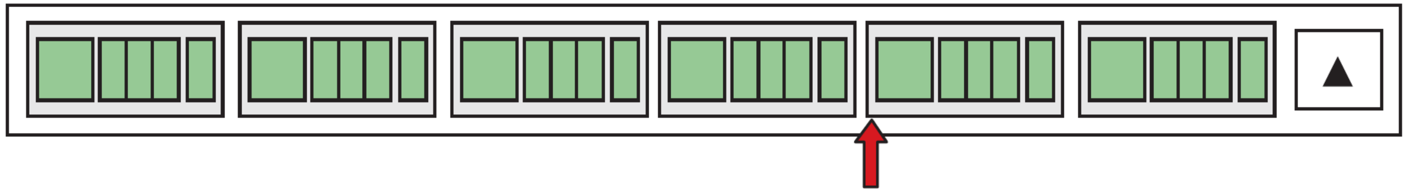


File Seek Operation(1/2)

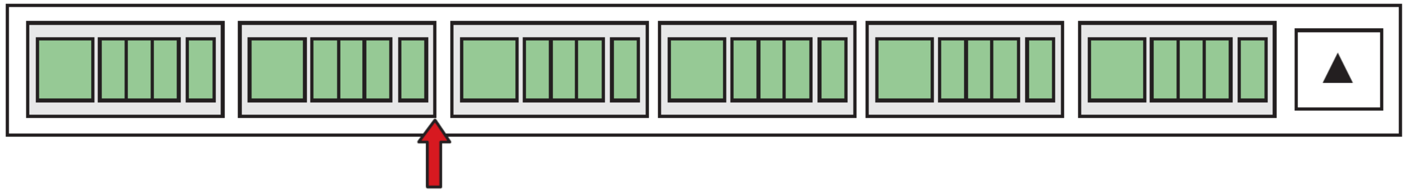
- Set position
  - int fseek(FILE* stream, long offset, int wherefrom);
  - Sets the file position indicator for *stream*
  - New byte position is obtained by adding *offset* to the position specified by *wherefrom*
  - *wherefrom*
    - SEEK_CUR: The offset is computed from the current position in the file
    - SEEK_SET: The offset is computed from the beginning of the file
    - SEEEK_END: The offset is computed from the end of the file

File Seek Operation(2/2)

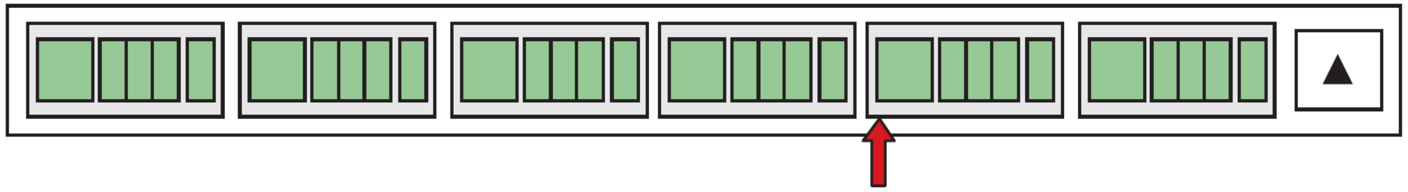




```
fseek (sp, 4 * sizeof(STRUCTURE_TYPE), SEEK_SET);
```



```
fseek (sp, - 4 * sizeof(STRUCTURE_TYPE), SEEK_END);
```



```
fseek (sp, 2 * sizeof(STRUCTURE_TYPE), SEEK_CUR);
```

File Status

- int feof(FILE* stream);
  - Checks the end-of-file indicator for *stream* and returns non-zero if it is at the end
- int ferror(FILE* stream);
  - Checks the error indicator for *stream* and returns non-zero if an error has occurred
- void clearerr(FILE* stream);
  - Clears the end-offile and error indicator for *stream*

Example: fseek before fwrite

```
#include <stdio.h>
#include <stdlib.h>
#include "student.h"
#define STRAT_ID 001

int main(int argc, char **argv)
{
    struct student record;
    FILE *fp;

    if (argc !=2)
    {
        fprintf(stderr,"Usage: %s filename\n", argv[0]);
        return 1;
    }
    fp = fopen(argv[1], "wb");
    printf("%s %7s %6s\n", "Sno", "Sname", "Sgrade");
    while (scanf("%d %s %lf", &record.id, record.name, &record.score) == 3)
    {
        fseek(fp, (record.id-STRAT_ID)*sizeof(record), SEEK_SET);
        fwrite(&record, sizeof(record), 1, fp);
    }
    fclose(fp);
    return 0;
}
```

Example: fseek before fread (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include "student.h"

#define STRAT_ID 001

int main(int argc, char **argv)
{
    struct student record;
    char c; int id;
    FILE *fp;

    if (argc !=2)
    {
        fprintf(stderr,"Usage: %s filename\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    if ( (fp = fopen(argv[1], "rb"))  == NULL)
    {
        fprintf(stderr, "Error: Cannot open the file %s\n", argv[1]);
        return 2;
    }
```

Example: fseek before fread (1/2)

```
    do {
        printf("Enter SNo for searching students> ");
        if (scanf("%d", &id) == 1)
        {
            fseek(fp, (id-STRAT_ID)*sizeof(record), SEEK_SET);
            if( (fread(&record, sizeof(record), 1, fp) > 0) &&
                (record.id != 0) )
            {
                printf("-------------------------------------\n");
                printf("%s\t %7s\t %6s\n", "Sno", "Sname", "Sgrade");
                printf("-------------------------------------\n");
                printf("%d\t %s\t %lf\n", record.id, record.name, record.score);
            }
            else printf("No such a student\n");
        } else printf("Input error\n");

        printf("Do you want search a student again? (Y/N) ");
        scanf(" %c", &c);
    } while( c == 'Y' || c == 'y' );

    fclose(fp);
    return 0;
}
```

Q&A